Cloud-Optimized Format Study

January 28, 2020

Prepared Under Contract NNG15HP39C

EOSDIS Evolution and Development - 2 Contract

RESPONSIBLE AUTHORS

Patrick Quinn, Principle Software Engineer

Chris Durbin, Principle Software Engineer

Dana Shum, Principle Software Engineer

Raytheon Company
Riverdale, Maryland

Table of Contents

Table of Contents ii

Introduction 1
- Purpose 1

Approach 1
- Summary of Formats Studied 1
  - Formats in Use 1
  - Network-Optimized Formats 3
- Methodology 7
  - Weighted Matrix 7

Evaluated Criteria - Weighted Matrix 8

Evaluated Criteria - Details 9
- Usability 9
  - Support for Fine-Grained Access 9
  - Support for a Variety of Data Types and Structures 9
  - Data Integrity 11
  - Self-describability 11
- Tooling and Standards 12
  - Compatibility with Existing Tools 12
  - Open Specification 13
  - Independent APIs 13
  - Programming Language Support 14
  - Standards-Body Approval 14
  - Ability to Comply with Metadata Conventions 15
- Cost Factors 15

Performance Benchmarking 15
- Methodology 15
  - File Types and Formats Used 16
  - Access Patterns Tested 16
  - Measurements 17
- Results 17
  - Benchmark Execution Time 18
  - Percentage of Bytes Accessed 20
  - File Size 22
- Usability Findings During Benchmarking 23
  - Lack of GDAL Support 23
  - Errors in NetCDF-4/ HDF Libraries 23

Necessary Performance Tuning for NetCDF-4/ HDF 24
Performance Influencing Usage Choices 24

Additional Considerations 24
File Packaging and Benchmarking 24
Server Software 25
Other Formats 25

Conclusions 27
Overarching Recommendations 27
Format Comparisons 27
Long-Term Archival Formats 27
Multispectral Data Formats 27
Multidimensional Array Data Formats 28
Specialized Data Formats 28
Recommended Follow-On Work 29

Appendix A: Acronym List 30

Appendix B - Scoring Justification and Notes 31

Introduction

Purpose

“The cloud infrastructure provides a number of capabilities that can dramatically improve access and use of Earth Observation data. However, in many cases, data may need to be reorganized and/or reformatted in order to make them tractable to support cloud-native analysis/access patterns. The purpose of this study is to examine the pros and cons of different formats for storing data on the cloud. The evaluation will focus on both enabling high-performance data access and usage as well as to meet the existing scientific data stewardship needs of EOSDIS.” (Task 51 Statement of Work, Revision B - Enabling EOSDIS Data Usage Statement of Work)

With the above purpose in mind, the evaluation we present seeks to inform future decisions on both archival formats for cloud data as well as formats to use for analysis, if different from the archival format. We do not expect to produce a recommendation for a single format for use in all cases but rather hope to inform format selection for a given use based on archival and usability needs. Further study may be warranted as formats, EO data, cloud infrastructure and user behavior evolve.

Approach

Summary of Formats Studied

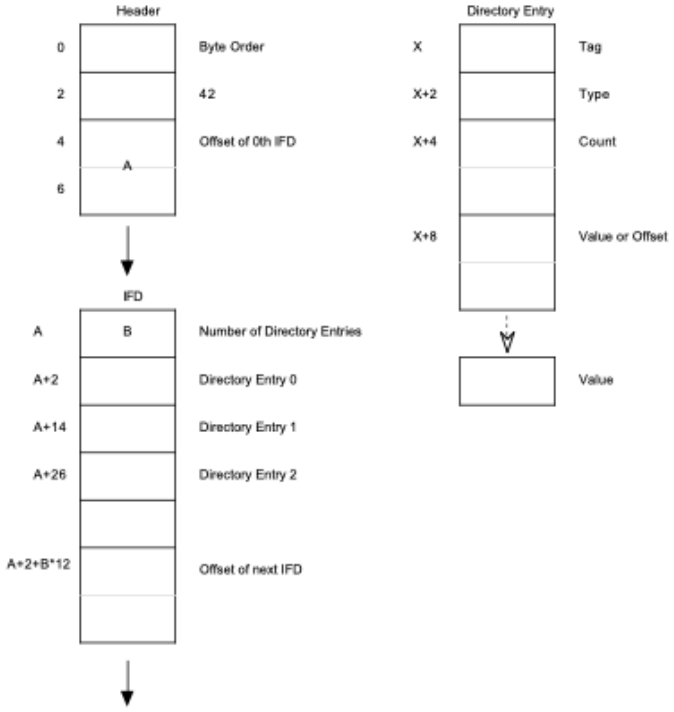
We examine several data formats that enable high-performance analysis for network-based access, particularly the cloud. We look at Cloud-Optimized GeoTIFF, HDF in the Cloud, Parquet, and Zarr, Parquet in depth. We then compare these to GeoTIFF and NetCDF-4, two existing standards which store the majority of current EOSDIS data. Although there are some server-based solutions that can provide similar network optimizations in the cloud (e.g., OPeNDAP in the Cloud and the Highly Scalable Data Server), they are out of scope for this study.

Formats in Use

GeoTIFF

GeoTIFF is an approved EOSDIS standard as well as an OGC standard built on the TIFF ISO standard. It stores layers of two-dimensional raster data and allows the expression of georeferencing information as well as other geospatial metadata. Though an image format, it is able to represent its data as arrays of numeric types ranging from bytes to 64-bit floating point numbers. The image below visually shows the TIFF specification.

Figure 1



NetCDF-4

NetCDF-4 is an approved EOSDIS and OGC standard. It is the encoding of the NetCDF-4Extended data model in the HDF5 file format, thus using HDF as the underlying storage format, with its programmatic API built on top of the HDF5 library. In qualitative comparisons, NetCDF-4 and HDF5 are very similar. As they constitute archival data standards in current use, we do not distinguish them for the purpose of this study. NetCDF-4 stores data in a hierarchical, array-based format. It is capable of describing detailed metadata on variables, units, georeferencing, and provenance.

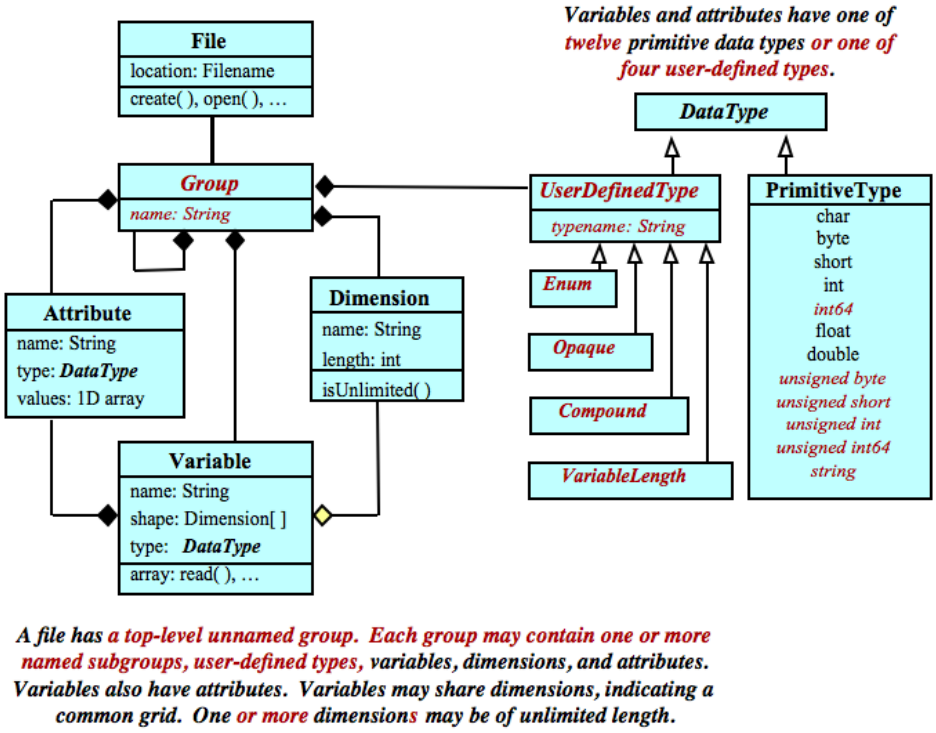


Image courtesy of UCAR/Unidata

Network-Optimized Formats

Cloud Optimized GeoTIFF

Cloud Optimized GeoTIFFs (COGs) are standard GeoTIFFs. They have an internal structure allowed, but not enforced, by the GeoTIFF standard. Because of this, we strongly recommend that GeoTIFFs for data products be cloud optimized, due to their significant benefits with virtually no drawbacks; in fact, tools such as GDAL produce Cloud Optimized GeoTIFFs by default when requesting GeoTIFFs because of this. The internal structure of COGs promotes efficient consumption of the format in networked environments, meaning that relatively few accesses of a portion of a COG using HTTP GET range requests can be used to extract metadata, overview imagery, a spatial subset, and / or a single multispectral band. Cloud Optimized GeoTIFFs also define a structure for a pyramid of overview tiles, allowing the possibility of low-bandwidth preview before access. The specification can be found at https://github.com/cogeotiff/cog-spec/blob/master/spec.md.

HDF in the Cloud

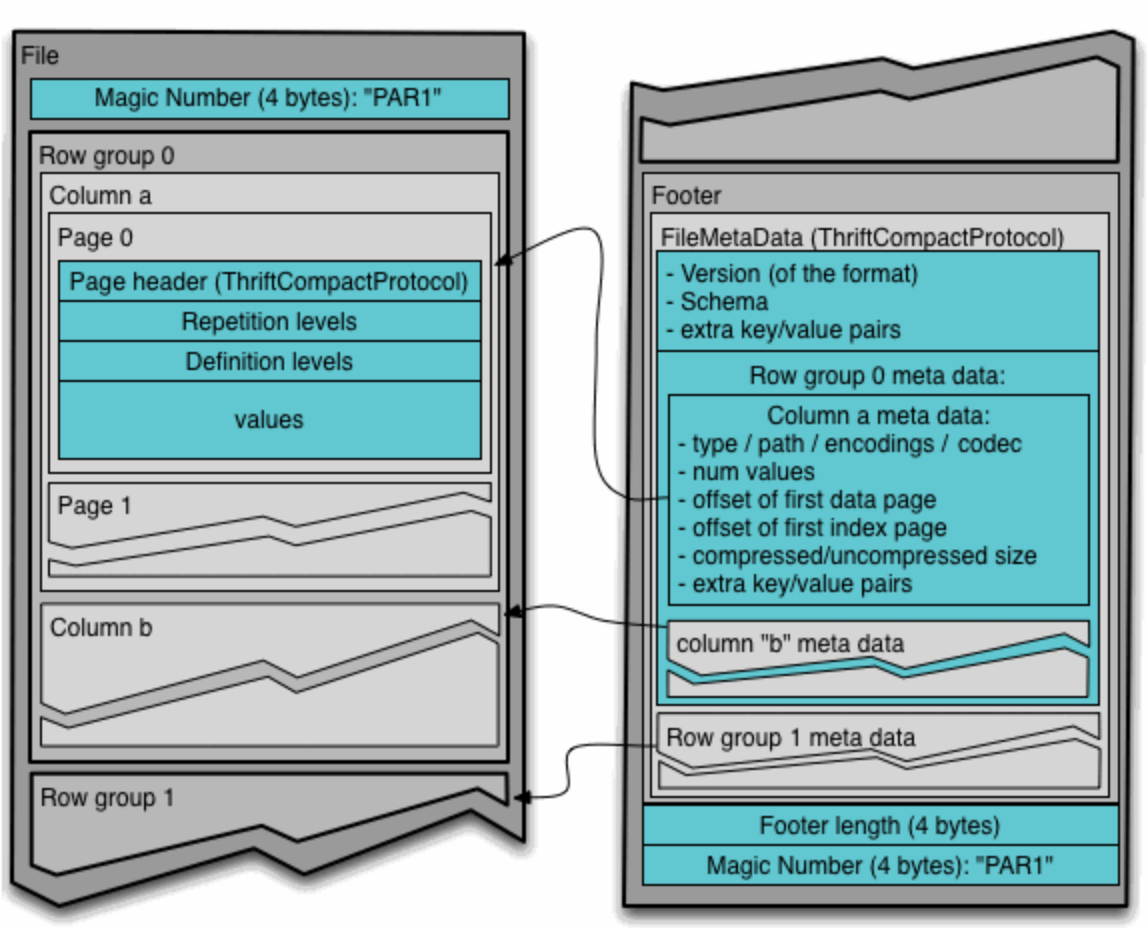
In this paper, the storage format of The HDF Group's Highly Scalable Data Service (HSDS) and commercial Kita offering is referred to as "HDF in the Cloud". This format consists of multiple objects in an object store, where each object is either JSON storage metadata or HDF5 dataset data chunks. The JSON metadata establishes the hierarchical organization of other stored HDF5 objects.

This HSDS object store format can also be used to provide a mapping to one or more traditional HDF5 files stored as objects. This enables HSDS to provide read capability to existing files by using HTTP range GET requests to read individual chunks from the file as needed.

When HSDS server runs alongside the object store, clients can use the HDF5 library APIs to read files as though they were stored on local disk. Importantly, though, this HDF5 compatibility does not mean that the actual stored objects are in the HDF5 file format. The HDF Group provides tools for bi-directional conversion between HDF5 files and the HSDS object store format. For the purpose of this study, we distinguish between the HDF5 file format and the HSDS server with client API (HDF5 REST). For example, despite the fact that many tools can use the HDF5 API with HSDS and the HDF5 REST Virtual Object Layer (VOL) connector, we are not aware of any tools that are able to read the HSDS storage format (HDF in the Cloud) directly. The HDF5 REST VOL is part of the latest HDF5 1.12.0 release. This is an important distinction for archival reasons, operational costs, and risk mitigation. For more discussion of this decision, see “Server Software” under “Additional Considerations.”

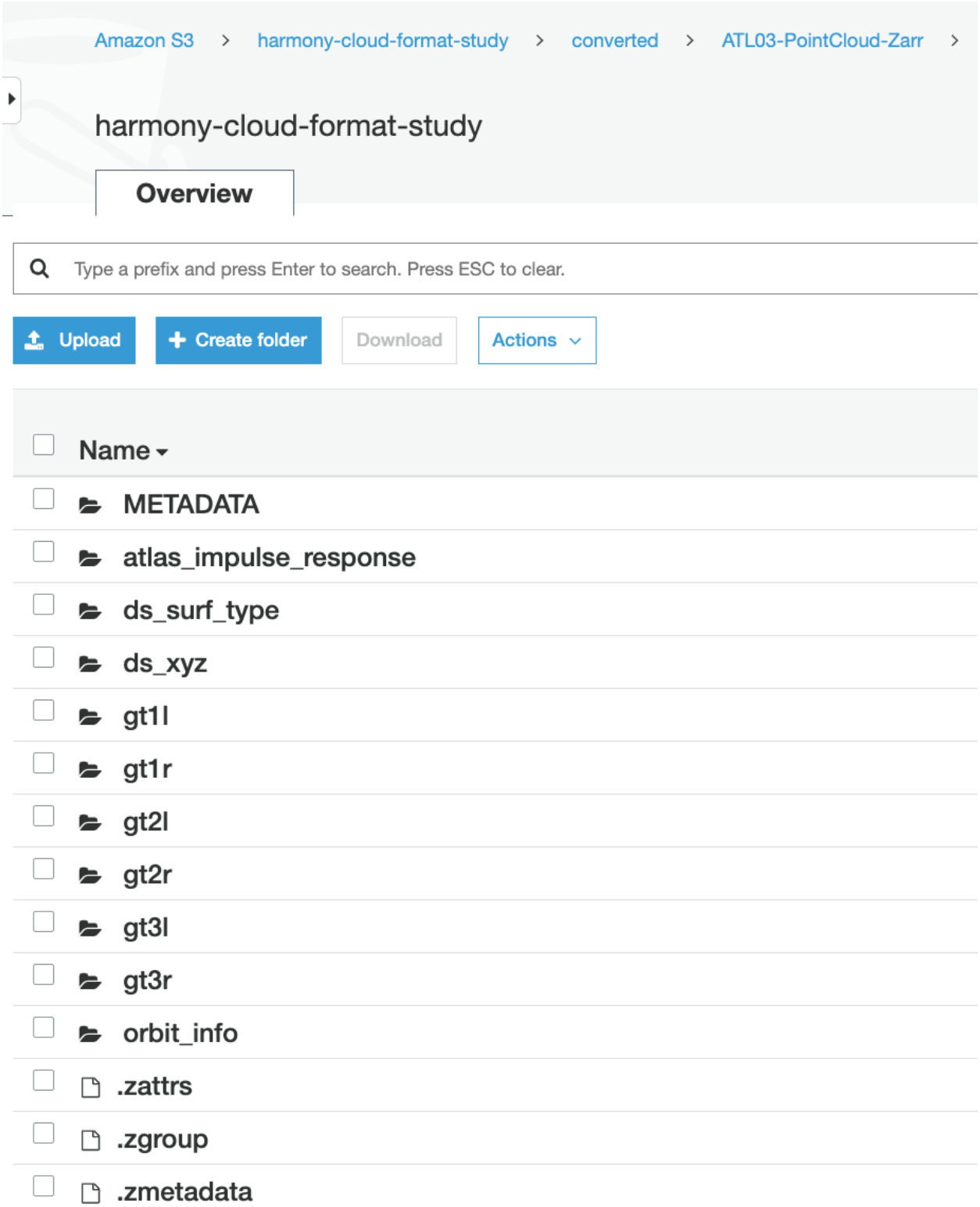
Parquet

Parquet is a columnar data store which is part of the Hadoop ecosystem and can be read natively by Amazon S3 Select queries and Apache Spark. It is able to capture rich metadata on a per-column basis and supports a wide array of data types, including custom ones. Being a columnar data store, it is best suited to situations where data can be expressed as a fixed-size tuple.

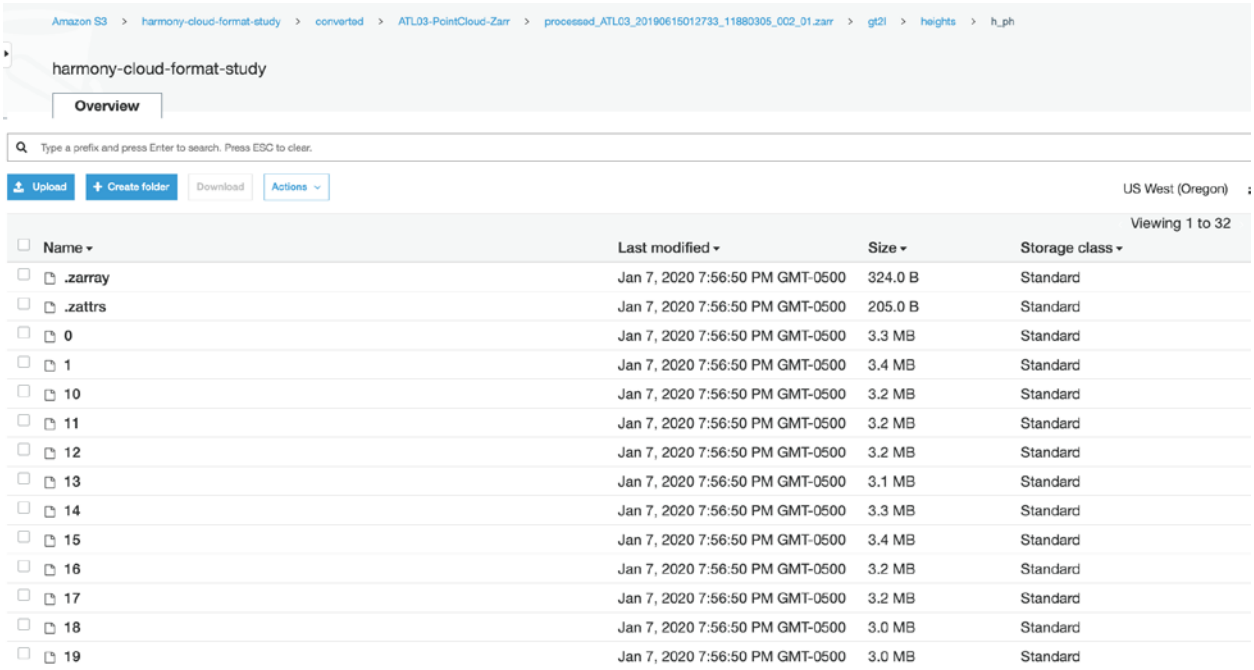


Zarr

Zarr is an array-based hierarchical data store, conceptually similar to NetCDF-4in terms of its ability to capture and express metadata and data. It stores hierarchies as “logical paths” which by default expand to separate keys in object stores such as Amazon S3, so a single conceptual file may in fact be several files on disk or in object stores. This can improve both parallelism and granularity in access.



An example ATL03 granule in a Zarr directory store. The .zmetadata file contains enough information for Zarr libraries to locate necessary files for any read operation performed. The directory and file structure mimic the variable information contained in the original NetCDF-4file.



Navigating to the files containing the variable data for the gtl1/heights/h_ph variable contains the chunked data for that variable. Each individual file from 0 to 36 contains a single chunk.

Methodology

Weighted Matrix

We compare the following criteria from our Statement of Work:

- Data access performance to support common forms of analysis, including time series, shape-based averaging, regridding and data intercomparison.
- Compatibility with existing off the shelf tools, including Panoply, gdal, nco, Jupyter/xarray, ArcGIS and QGIS.
- Ability to support fine-grained requests from S3 via range-get or other means.
- Ability to comply with community metadata conventions (e.g., CF)
- Availability of independent libraries to read the data in C/C++, Fortran, Python and R
- Comparative cost of data preparation, storage and analysis, adjusted for lossless compressibility as appropriate.
- Ability to represent several different data types / structures including imagery, swath, trajectory, point cloud, Platte-Carre and Sinusoidal grids, in situ and airborne
- Ability to verify data integrity upon reformatting and ongoing
- Self-describability, i.e., ability to include complete sets of both descriptive and structural metadata
- Open specification

- Number of independent implementations of read/write API
- Standards-body approval (OGC, W3C, etc.)

For each criterion, we assign a score to each format, 0 through 9, providing both a description of each score and a justification for why each format received the score. We do not assign weights to the criteria but do provide a spreadsheet in Appendix B which others can assign weights appropriate to their needs.

We additionally synthesize findings in this paper, describing the outcomes and also noting circumstances where scores may be more or less important, e.g. a format could score very low in one criterion for only representing one data type, but it may be particularly fast, inexpensive, or useful when the data in question are of that type.

Evaluated Criteria - Weighted Matrix

		Current Formats		**Cloud-Optimized Formats**			
Criteria	**Criteria Weight**	**netCDF**	**GeoTIFF**	**Zarr**	**Parquet**	**Cloud - Optimized GeoTIFF**	**HDF in the Cloud**
Support for Fine-Grained Access	1	3	3	6	9	9	6
Support for a Variety of Data Types and Structures	1	9	3	6	0[1]	3	9
Data Integrity	1	9	3	3	0	3	6
Self-describability	1	9	6	9	9	6	9
Compatibility with Existing Tools	1	3	6	1	1	6	0
Programming Language Support	1	9	9	3	6	9	0
Open Specification	1	6	6	6	9	9	6
Independent APIs	1	3	6	3	9	6	0
Standards-Body Approval	1	6	9	0	3	9	3
Ability to Comply with Metadata Conventions	1	9	3	9	6	3	9
Weighted Score		**66**	**54**	**46**	**52**	**63**	**48**

[1] While Parquet is able to represent a number of different data formats and complex data structures, it is generally unaware of the spatiotemporal nature and structure of the data. It has no issue representing individual data points, but it loses some of the internal structure and locality of reference relating points to one another.

Evaluated Criteria - Details

Each section below explains both the qualitative and quantitative analysis related to each criteria. This information determines the score for each item in the Weighted Matrix above.

Usability

Support for Fine-Grained Access

For fine-grained analysis, we looked at each format's inherent ability to accommodate accessing small areas of interest within a file without having to download a large percentage of the overall data file, e.g. using HTTP Range GET requests. Unsurprisingly, the cloud-optimized formats all fare well in this category, as it is a primary facet of what it means to be cloud-optimized. Existing formats do not necessarily fare poorly, however, but they are highly dependent on the underlying structural choices of the data. In the best circumstances, a GeoTIFF could be cloud optimized[2] and a near-future NetCDF-4file could be stored as Zarr[3,4] In the worst, they could be chunked and compressed in such a way that a download of the entire file followed by local processing would be significantly faster than network processing. Because there is no standard way to indicate to users or tools that GeoTIFF or NetCDF-4file is optimized for network access (vs disk access or total size), documentation and examples will need to supplement such files accessed in the cloud in order to assure the lowest cost and highest performance.

Support for a Variety of Data Types and Structures

The investigated file formats have varying degrees of support for different data structures, ranging from generalists that support a wide range of structures to specialists that optimize access to one type of data. That, indeed, is a key tradeoff: specialized formats benefit from being able to optimize their use case and target their tooling, while generalized formats are able to handle many more types of data but have less a priori knowledge of their own structure to aid in optimization or focused tooling.

Parquet and GeoTIFF (including Cloud-Optimized GeoTIFF) provide storage specialized to particular data types. Parquet is suitable for columnar data of the type often found in a database table or spreadsheet[5], with the advantage of providing fast database-style queries using both Hadoop and S3 Select. GeoTIFF is most suitable for two-dimensional imagery and

[2] "A Cloud Optimized GeoTIFF (COG) is a regular GeoTIFF file"
"Cloud Optimized Geotiff," accessed November 13, 2019, https://www.cogeo.org/.

[3] Starting with the HDF5 version 1.10.6 released in December 2019, new functionality was added to the HDF5 library to access HDF5 files in S3 store using HTTP range GET requests. While access is not fully optimized at this point one doesn't need to download the whole file to access areas of interest. New optimizations are planned for the next HDF5 releases that will minimize the number of accesses to HDF5 internal structural metadata that includes, for example, locations of raw data chunks and user-defined attributes thus speeding up access to the data of interest

[4] Ward Fisher, "NetCDF and Native Cloud Storage Access via Zarr," *News @ Unidata*, last modified June 12, 2019, accessed November 13, 2019, https://www.unidata.ucar.edu/blogs/news//entry/netcdf-and-native-cloud-storage.

[5] Apache/Parquet-Format (The Apache Software Foundation, 2019), accessed November 13, 2019, https://github.com/apache/parquet-format.

benefits from the tool and library support of the decades-old open format it is built upon[6]. While one could potentially express different data types, in either file format, doing so would diminish the benefits of choosing that format. For example, one could inflate a multidimensional array into a columnar format of (x, y, z, …) coordinates for Parquet, but doing so would simultaneously increase the storage size and lose the spatial structure inherent in other formats. We note that the ease of querying information in this format may still make Parquet compelling, however.

On the other end of the spectrum are the more general-purpose NetCDF, HDF5, and Zarr formats. They handle multidimensional arrays and hierarchical data, allowing them to accommodate many different types of data. NetCDF, HDF5, and Zarr are known to be combined with the Climate and Forecast Metadata Conventions which enable them to be highly suitable for vast amounts of geoscience data types Their expressiveness, however, necessitates more care on the part of file authors to ensure that data are structured in a manner that is performant for their intended use.

Below is a table of the various formats and their suitability for each data type.

	netCDF	GeoTIFF	Zarr	Parquet	Cloud-Optimized GeoTIFF	HDF in the Cloud
Imagery	Moderate	High	Moderate	Poor	High	Moderate
Swath	Moderate	Low	Moderate	Low	Low	Moderate
Trajectory	Moderate	Poor	Moderate	Moderate	Poor	Moderate
Point Cloud[7]	Low	Poor	Low	Moderate	Poor	Low
Projected Grids	High	High	High	Low	High	High
In Situ Data	Moderate	Poor	Moderate	High	Poor	Moderate
Airborne Data	Moderate	Low	Moderate	Low	Low	Moderate
Vector + values	Poor	Poor	Poor	High	Poor	Poor

Key:

High	Specialized support
Moderate	Supported
Low	Possible to express with some interpretation of the format and/or compromises in space and performance
Poor	Requires significant compromises in space or performance or otherwise not possible to express

[6] Geospatial World, “GeoTIFF - A Standard Image File Format for GIS Applications,” Geospatial World, September 10, 2009, accessed November 13, 2019, https://www.geospatialworld.net/article/geotiff-a-standard-image-file-format-for-gis-applications/.

[7] While point clouds are frequently represented in array-based formats such as NetCDF, we give this a “low” level of support based on algorithmic complexity for common operations. Locating the nearest neighbor for a given point requires a full read of the data for array-based stores but would be a logarithmic operation in a k-d tree representation. None of the evaluated formats have this sort of specialized support for point data.

Data Integrity

For data integrity, we look at each format's ability to have its data validated. The ideal in this category would be a format that captures fine-grained (per-chunk, band, or layer) integrity checking and is able to preserve integrity checking after altering the internal file arrangement so long as data is not altered. While no formats landed fully in this category, NetCDF-4accessed through HDF5 and HDF in the Cloud come the closest, due to its ability to capture fine-grained integrity information and the presence of the h5check[8] command-line tool checks compliance with the HDF5 file format specification..

Regardless of it not reaching the ideal, NetCDF-4attained the most points of any format in this category, as it maintains internal integrity checking information[9].

Zarr and the two GeoTIFF formats received the next highest points, as neither has a consistent way to provide integrity checks within the file, though we note that each could do so with metadata fields, as could Parquet or any format allowing application-specific metadata. Zarr's multi-file key/value storage model[10] is conducive to finer-grained integrity checking and there is community discussion surrounding including integrity information in upcoming versions. GeoTIFF allows fine-grained external integrity checking by, for example, running gdalinfo with the "-checksum" flag.[11]

Parquet receives the lowest score in this category, as it neither supports internal file integrity checks (though again, this could be done with metadata fields) nor does the format support fine-grained integrity checking in any particular way.

Self-describability

All of the investigated formats are self-descriptive in terms of their structure. All of them also provide some provision for application-specific metadata. We note that file formats without a long history of use for archiving Earth observation data may need additional conventions to be built regarding how, e.g., provenance fields are stored, whereas NetCDF-4and HDF already have these conventions.[12]

The only score variance here is GeoTIFF (including Cloud-Optimized GeoTIFFs). Because GeoTIFF may require or encourage application metadata to be placed in a sidecar PAM file if it is large[13], GeoTIFFs may not be as self-descriptive as the other formats in practice. The fact that it remains structurally self-descriptive, that it is theoretically capable of describing application metadata, and that there are conventions around the existence and placement of the PAM files caused us to not greatly penalize the format in this category, however.

[8] See https://support.hdfgroup.org/products/hdf5_tools/h5check.html for details and limitations

[9] By e.g. Fletcher32 checksumming variables. See https://www.mathworks.com/help/matlab/ref/netcdf.defvarfletcher32.html

[10] See https://zarr.readthedocs.io/en/stable/spec/v2.html#storage

[11] See https://gdal.org/programs/gdalinfo.html

[12] See http://cfconventions.org/

[13] "gdal/geotiff.cpp · OSGeo/Gdal," GitHub, accessed November 13, 2019, https://github.com/OSGeo/gdal/blob/7d9482a63266c5ceb174f0ccfd4c23a10b258bde/gdal/frmts/gtiff/geotiff.cpp#L11690.

Tooling and Standards

Compatibility with Existing Tools

We looked at interoperability with the following common tools and libraries:

- Panoply
- gdal
- nco
- Jupyter / xarray
- ArcGIS
- QGIS
- OPeNDAP

In each instance, we only looked at current (as of the time of this writing) support for the format either through the tool's binary distribution or readily available plugins.

For each tool, we gave three points if the tool could read the format over a network with only partial downloads, one point if the tool could read the format only after a full download, and no points if the tool could not read the file. The former two number weights are configurable in the accompanying spreadsheet referenced in Appendix B. We then normalized the resulting points on a scale from 1 to 9.

For the tools investigated, NetCDF-4has universal support, but also universally requires either a full-file download before reading the files or an OPeNDAP server in front of the files to allow partial reads. The prevalence of tool compatibility with OPeNDAP makes the case for NetCDF-4much stronger than it would be for NetCDF-4alone, though we have no metrics as to what percentage of users discover and use OPeNDAP when a full-file download is available.

GeoTIFF has the most support of network-based access of any of the formats with the tools investigated. Only two of the tools listed do not support reading the format (Panoply and nco), both of which were purpose-built for working with NetCDF.

Parquet and Zarr both currently focus on strong support for particular software, having matured alongside Hadoop and the Zarr Python library, respectively. Neither has particularly strong tool support among those listed, however, Parquet has a notable advantage of being supported by S3 Select queries, and Zarr will be readable from the NetCDF-4API in the future.

The notable outlier in this comparison is HDF in the Cloud. Currently the only software that is able to read HDF in the Cloud is The HDF Group's Highly-Scalable Data Service (HSDS)[14] and their proprietary Kita server.[15] While many clients, such as xarray, appear to be able to read data exposed by HSDS through the HDF5 REST API using the h5pyd Python package, this is not a fair comparison since, for example, we are not factoring in how many clients would be able to read GeoTIFFs exposed by GeoServer[16] using WMS or how many clients would be able to read NetCDF-4(HDF5) files exposed by Hyrax[17] using OPeNDAP. This comparison only covers file formats not file formats plus server software.

[14] https://github.com/HDFGroup/hsds
[15] https://www.hdfgroup.org/solutions/hdf-kita
[16] http://geoserver.org/
[17] https://www.opendap.org/software/hyrax-data-server

Open Specification

All of the formats we investigated have open specifications. We compare governance of the specifications in "Standards-Body Approval." For this section, we looked at facets of the specification documentation that would impact an adopter's ability to understand, troubleshoot, and if necessary produce a compatible tool to read the standard's data, all of which impact the viability of the standard as an archival format.

Importantly, this meaning must be factored into the weight given to this category. While not being an open standard may be disqualifying for a format, none of the formats fall into that category. The distinctions we make in this category come down to documentation and specificity, which are more nuanced distinctions.

All formats fared well in this category, with documentation that is specific and formal enough to reproduce a reader for the format if necessary. Both Parquet[18] and Cloud Optimized GeoTIFF[19] fared slightly better, Parquet due to its clear diagrams and descriptions to aid in a thorough understanding of the format, and Cloud Optimized GeoTIFF due to it having an associated test script to validate conformance to the standard.[20]

Independent APIs

A key question for each of the formats evaluated is how dependent a format is on a specific library implementation. We looked into the number of independent libraries for reading and writing for each format. In addition we considered if the main library used for the format were to disappear, how difficult would it be to implement another library in its place.

All libraries implementing reads and writes for HDF5 files utilize the same underlying libraries provided by the HDF Group. For the HDF in the cloud format the libraries that are used all rely on an HSDS server in order to perform reads. There is a JSON metadata file stored along with the inflated HDF file contents in S3 that should make it possible to write an independent library to read the file, however the same cannot be said for writes. The NetCDF-4 format also uses one main implementation for reading and writing files provided by Unidata.

GeoTIFF files mainly utilize the libgeotiff library for performing reads or writes, but other libraries have been built which can read GeoTIFF[21]. The TIFF standard on which GeoTIFF is built has many independent implementations, and any conformant TIFF implementation can read GeoTIFF though likely will not interpret its metadata tags.

Zarr also provides one main library supporting reads and writes for the Zarr format. There is active development with the start of independent implementations and an evolving open file format specification, which increases the likelihood of multiple independent libraries available in the future.

---

[18] See documentation at https://github.com/apache/parquet-format

[19] See documentation at https://github.com/cogeotiff/cog-spec/blob/master/spec.md

[20] https://github.com/OSGeo/gdal/blob/master/gdal/swig/python/samples/validate_cloud_optimized_geotiff.py

[21] Pure Java implementation by UCAR: http://rhinohide.org/rhinohide.cx/co2/spatial-analyst/scripts/GeoTiffDataReader.java
Apache Commons Imaging: https://commons.apache.org/proper/commons-imaging/index.html

The Parquet format has the most independent implementations of all of the file formats. Apache provides the main Java and C++ implementations, but there are independent implementations in other languages including Python, Go, Rust, and many others. It is an open standard and includes a parquet-compatibility project, which can be utilized to verify an implementation.

Programming Language Support

We evaluated each of the formats based on programming language support for C, C++, Fortran, Python, and R.

NetCDF, HDF5, and GeoTIFF have libraries to support reading and writing in all five of the languages. However, as mentioned previously HDF in the Cloud does not support directly accessing the data in S3 without using an HSDS server, so the format itself does not have programming language support.

Parquet has direct library support for each of the languages with the exception of Fortran. However, given that Parquet has a C library, Fortran can still be used with C interop.

Zarr was the main outlier for programming language support. Zarr was initially developed using Python and the community is starting to add support for other languages. C++ is fully supported, and there is extensive discussion of adding C support with a library currently in development. There's also the start of a library for R though it is far from complete. There are no discussions regarding a Fortran library, which would again mean that Fortran developers would need to use C interop once the C library is ready or use the NetCDF-4API once it is adapted to support Zarr.

Standards-Body Approval

For this category, we looked at standards body approval, but in the absence of that approval, we additionally looked for a clear governance model for the standard.

GeoTIFF was a stand out in this category, as it is not only an OGC standard,[22] but it is built on and compatible with TIFF, which itself is an ISO standard[23] that has been unchanged apart from metadata and representation conventions since 1992, being vetted in software readers present in nearly every consumer operating system since then. As all Cloud Optimized GeoTIFF files adhere to the GeoTIFF standard, we consider them to have the same level of standards body approval. NetCDF-4is also an OGC standard format[24].

Neither Parquet nor HDF in the Cloud have standards body approval, but both have clear owners and stewards in the Apache Foundation and HDF Group respectively.

Zarr does not have any clear ownership or governance model[25]. It is community-developed and owned, and, while its contributing guidelines note that the maintainers will be conservative

[22] https://www.opengeospatial.org/standards/geotiff

[23] https://www.iso.org/standard/34342.html

[24] Although a public announcement by OGC is pending, the 18-043r3 OGC Hierarchical Data Format Version 5 (HDF5®) Core Standard has passed the entire OGC standard adoption process. Contact: Scott Simmons OGC Chief Operations Officer (ssimmons@ogc.org)

[25] Zarr is "Copyright (c) 2015-2018 Zarr Developers" https://github.com/zarr-developers/zarr-python/blob/ebf67f59bacabc50d0a2361a064b472730e6b1a4/LICENSE#L3 with governance documentation that is empty at the time of this paper https://github.com/zarr-developers/governance/tree/1196116411f542d7cfe4879c947ccfee77b5b999

in accepting changes, its criteria for incorporating change and stability are unspecified. Based on this, it receives the lowest possible score.

Ability to Comply with Metadata Conventions

In this category, NetCDF-4(and HDF by extension) is a clear choice based on the fact that CF conventions were built to work with the format[26] and it has ample additional metadata fields relevant to the Earth Science domain. Zarr has fields that allow capturing CF-compliant metadata and has plans to be readable by the NetCDF-4API in the future, allowing it to fully and compatibly expose those fields in the future.

Parquet has adequate fields to store metadata on variables and units[27], however we can find no accepted convention for the naming of those fields, nor any widespread community use, so existing conventions would need to be extended to the format.

GeoTIFF and Cloud Optimized GeoTIFF fare the worst in this category. While they support named layers, which could follow existing conventions, they do not provide a widely-adopted way to express unit information for individual layers. Further, GeoTIFF metadata is capped at a relatively low size (32000 bytes for those produced by GDAL[28]) necessitating offloading the metadata to sidecar "PAM" files which need to be distributed separately from the data file itself. This can make it relatively difficult to deliver metadata to users even when it is present.

Cost Factors

The "Performance Benchmarking" portion of this study describes specific factors that impact cost, including relative data sizes for each format, time spent performing various operations against data, and the amount of data accessed for those operations. While this is a significant step toward an overall cost picture, we choose not to assign precise dollar figures due to ill-defined operational costs and egress costs associated with accessing data that was processed in the cloud. Numbers not including these factors (in the accompanying spreadsheet in Appendix B) reveal that cloud-optimized formats do generally cost less to process and / or distribute than similarly-sized non-optimized formats, often significantly so. Of potentially more interest, we note that with some use cases and data types, the cost of processing in the cloud may be substantially more than the cost incurred by allowing users to directly access the data in an optimized format. This is not universal, but it is significant enough that it merits more detailed analysis when producing specific data products and services.

Performance Benchmarking

Methodology

After evaluating qualitative criteria, we perform a set of benchmarks to illuminate high-level differences in operational characteristics between netCDF, GeoTIFF, Cloud-Optimized

[26] "CF Conventions Home Page," accessed November 13, 2019, http://cfconventions.org/

[27] See column metadata: https://github.com/apache/parquet-format#file-format

[28] "gdal/geotiff.cpp · OSGeo/Gdal," GitHub, accessed November 13, 2019, https://github.com/OSGeo/gdal/blob/7d9482a63266c5ceb174f0ccfd4c23a10b258bde/gdal/frmts/gtiff/geotiff.cpp#L11690

GeoTIFF, and Zarr stored in AWS S3 and accessed both from an AWS and a non-cloud environment.

Producing benchmarks which exhaustively compare both use cases and data collections for each file would be infeasible, so we chose to focus on four high-level access patterns, across four common data types.

File Types and Formats Used

We used the following file types, with the specified formats for each type

- Gridded with global coverage where the grid aligns to latitudes and longitudes (3IMERGHH), NetCDF-4(native), Zarr, and Cloud-Optimized GeoTIFF
- Gridded with tile coverage where the grid does not align to latitudes and longitudes (MOD11A2), GeoTIFF (Converted from native by HEG) and Cloud-Optimized GeoTIFF
- Two-dimensional swath (AVHRR19_G-NAVO), NetCDF-4and Zarr
- One-dimensional point cloud (ATL03 v209), NetCDF-4and Zarr

We obtained and staged several (90 to 912) granule files from each collection above in Amazon S3. For our tests of NetCDF-4and non-optimized GeoTIFF, we performed no transformation or optimization on the source file as obtained from the archive.

We then converted each granule file into the desired optimized formats and staged those in Amazon S3. When converting NetCDF-4to Zarr, we used a library[29] which preserved the hierarchical format and metadata of the original. Where possible, we retained the chunk sizes of the original file. We used the default Zarr compression, Blosc. We found it necessary to consolidate the Zarr file's metadata for performance. This operation, with a convenience method in the Zarr API, traverses the Zarr hierarchy and produces a top-level file containing the metadata of each sub-directory and file.

When converting from GeoTIFF to COG, we used the one-line gdal_translate command provided in the COG developer's guide (https://www.cogeo.org/developers-guide.html).

As described in our methodology, each test we ran produces a numeric result that ensures the data read is consistent between formats, and indeed this illuminated a bug in our NetCDF-4processing for AVHRR19_G-NAVO that we were able to resolve.

Access Patterns Tested

While we chose realistic, meaningful use cases where possible, we focused on realistic byte access patterns rather than suitability for any scientific use. We compared the following operations for each file type and format.

- Report the value of a single point at a consistent array index within each file, slightly off-center. This amounts to a time series for grid-aligned data, given our file organization. For swath and point cloud data, this pattern is not as useful but we run it nonetheless to provide more data points on single-index access.
- Average points across a spatial subset for a parameter in each input file. We chose slightly irregular spatial subsets so that many useful data bytes could be fetched together but they wouldn't constitute a simple rectangle within the file's representation.

[29] https://github.com/bilts/netcdf-to-zarr

We averaged them and reported the answer to ensure the data had been read fully and correctly.

- Grid or re-grid each input file. Report the average of all the output points for validation. For swath and point cloud files, align the data to a 500x500 pixel plate carree / WGS84 grid using nearest neighbor interpolation. For gridded files, project the input file to a Mollweide projection centered on Greenwich.
- (Gridded only) Compare a parameter across two input files and produce a resulting array that contains the maximum valid value at each input point. Report the average of all the output points for validation.

Wherever possible, the benchmarking code used identical configuration, algorithms, and code paths, with the only distinction being the format driver. We did this using xarray as an abstraction layer over the underlying zarr, rasterio (GeoTIFF), and h5py (HDF5 / NetCDF) libraries.

Measurements

For each access pattern executed on each file type and format, we measure the following:

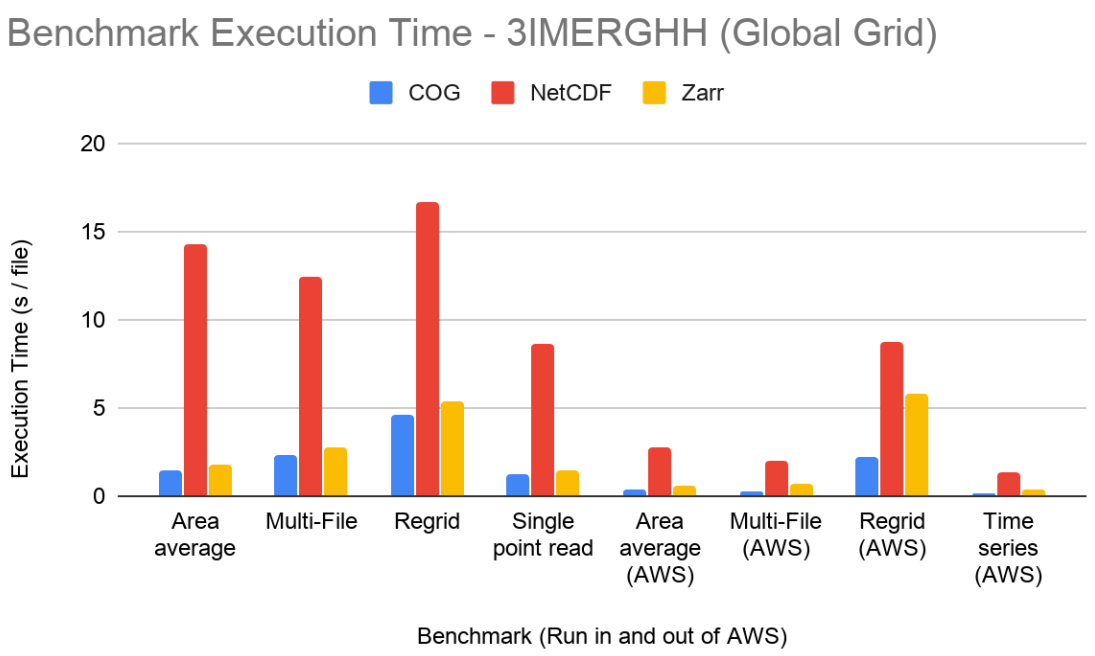
- Time to perform the operation from a c4.xlarge instance (4 vCPUs and 7.5 GB of memory) running in AWS in the same region as the data. This instance type provides a reasonable balance of CPU performance, network performance, and price. This simulates in-cloud services and usage.
- Time to perform the operation from a computer (2.8GHz Core I7 MacBook Pro with 16GB of RAM) fetching the data across the Internet. This simulates non-cloud usage.
- Number of requests made to Amazon S3 in performing the operation, which has both a cost and performance impact.
- Number of bytes transferred from Amazon S3 in performing the operation, which has both a cost and performance impact.
- Number of bytes consumed by the data staged in Amazon S3, which impacts cost.

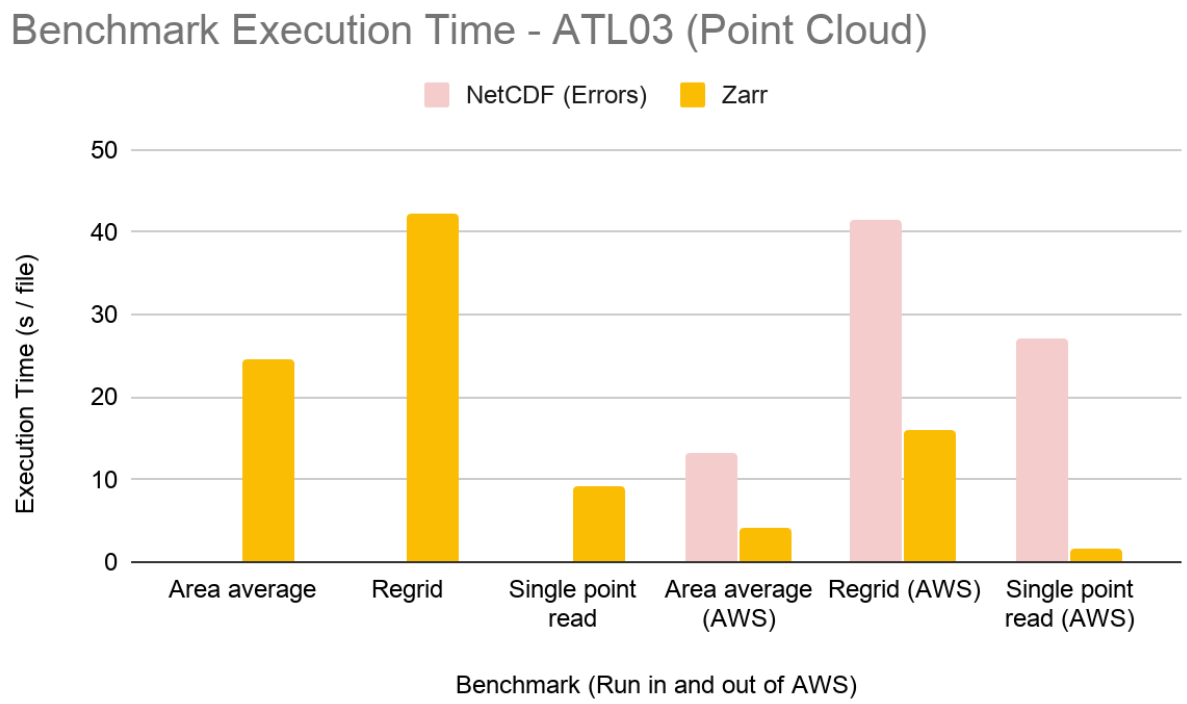
We further note any challenges encountered with file formats or libraries when performing the tests for qualitative comparison.

Results

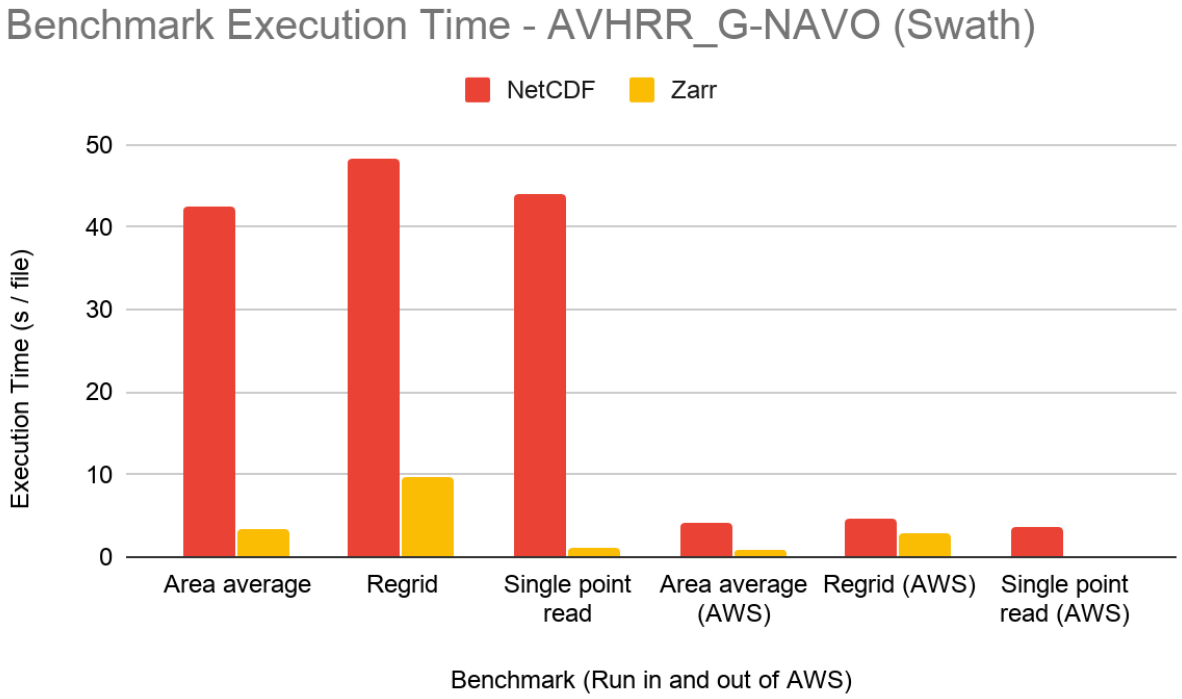
Note: The full numeric quantitative results are available in a spreadsheet linked in Appendix B.

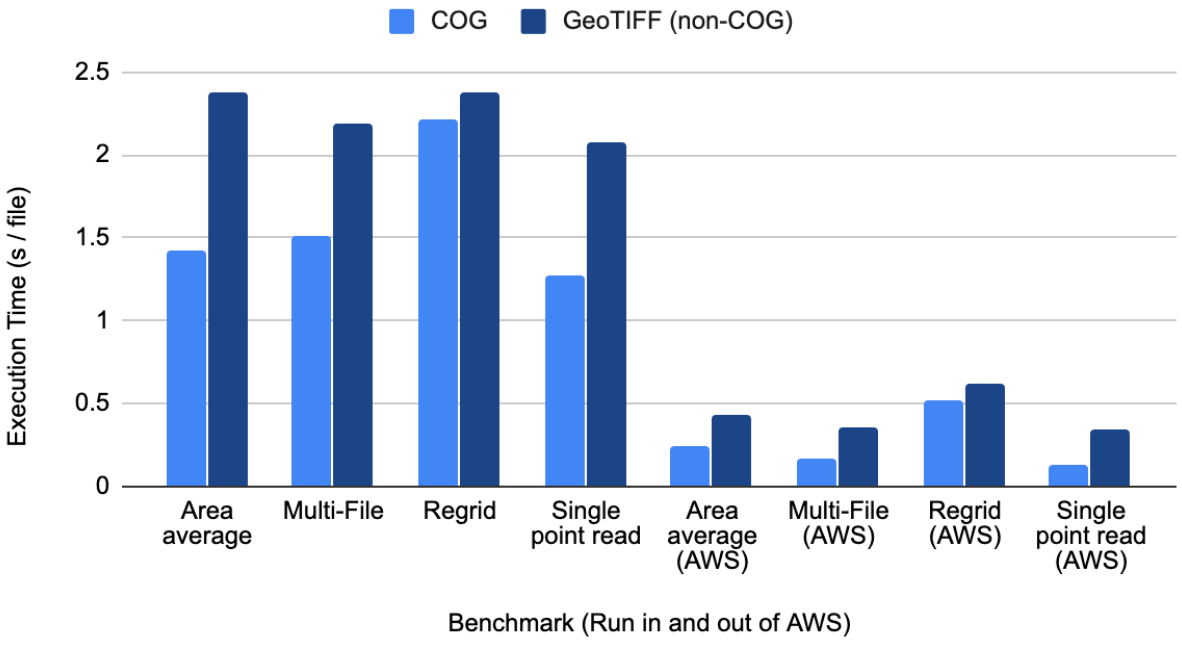
Benchmark Execution Time





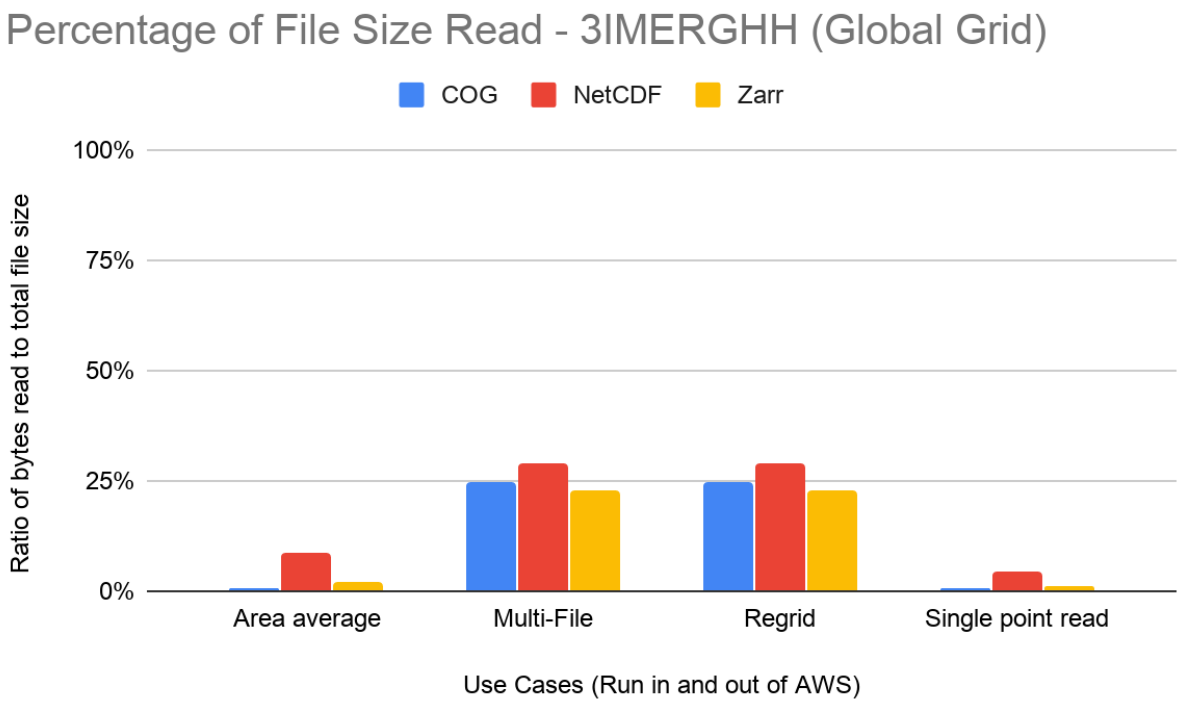
Note that for ATL03, we encountered errors in attempting to process HDF5 data in situ which did not occur when downloading the data. The above time represents the time to download the entire dataset within AWS plus a small delta for processing. We did not attempt full downloads for non-cloud access.

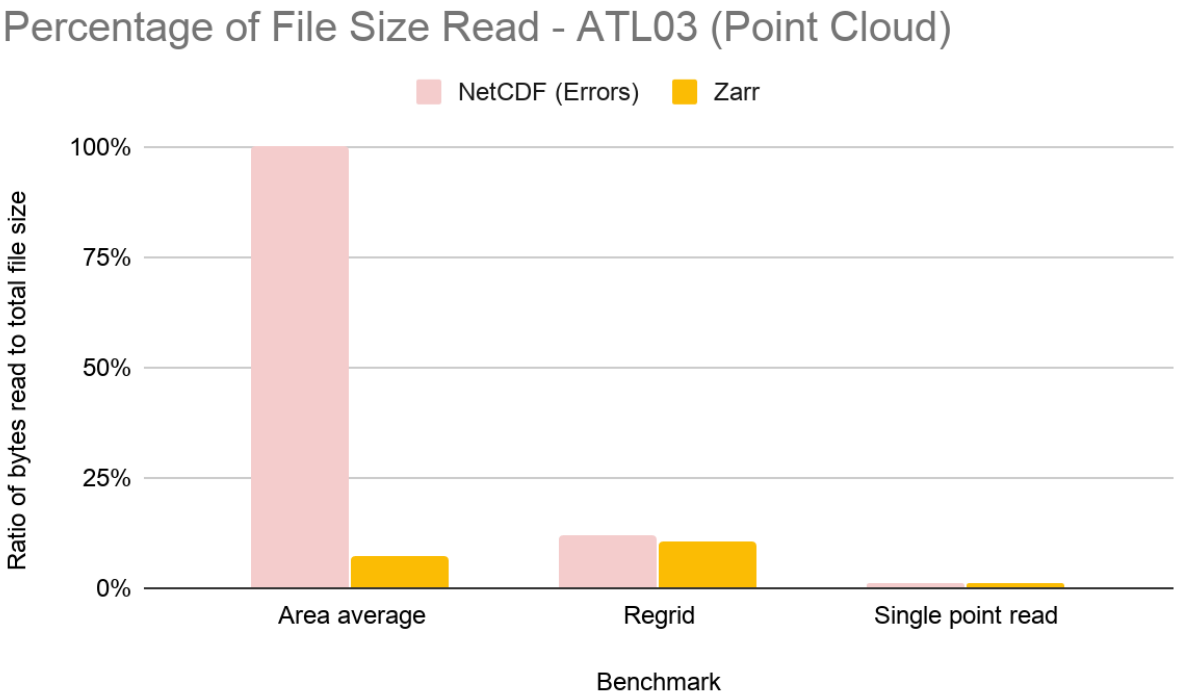

Benchmark Execution Time - AVHRR_G-NAVO (Swath)
NetCDF
Zarr
50
40
30
20
10
0
Execution Time (s / file)
Area average
Regrid
Single point read
Area average (AWS)
Regrid (AWS)
Single point read (AWS)
Benchmark (Run in and out of AWS)


Benchmark Execution Time - MOD11A2 (Tile Grid)
COG
GeoTIFF (non-COG)
2.5
2
1.5
1
0.5
0
Execution Time (s / file)
Area average
Multi-File
Regrid
Single point read
Area average (AWS)
Multi-File (AWS)
Regrid (AWS)
Single point read (AWS)
Benchmark (Run in and out of AWS)

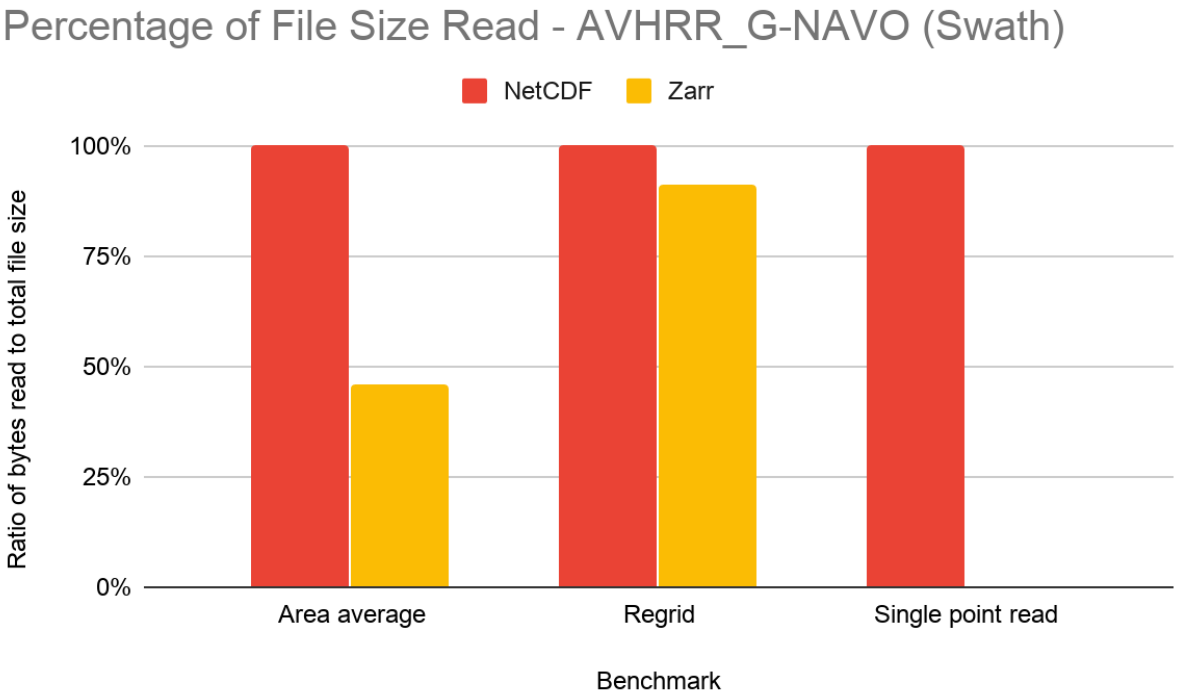
Percentage of Bytes Accessed

To better understand the overall access patterns and egress costs associated with in situ data processing, we report the number of bytes accessed as a percentage of the total file size, where 100% means the entire file was read in order to perform the processing.

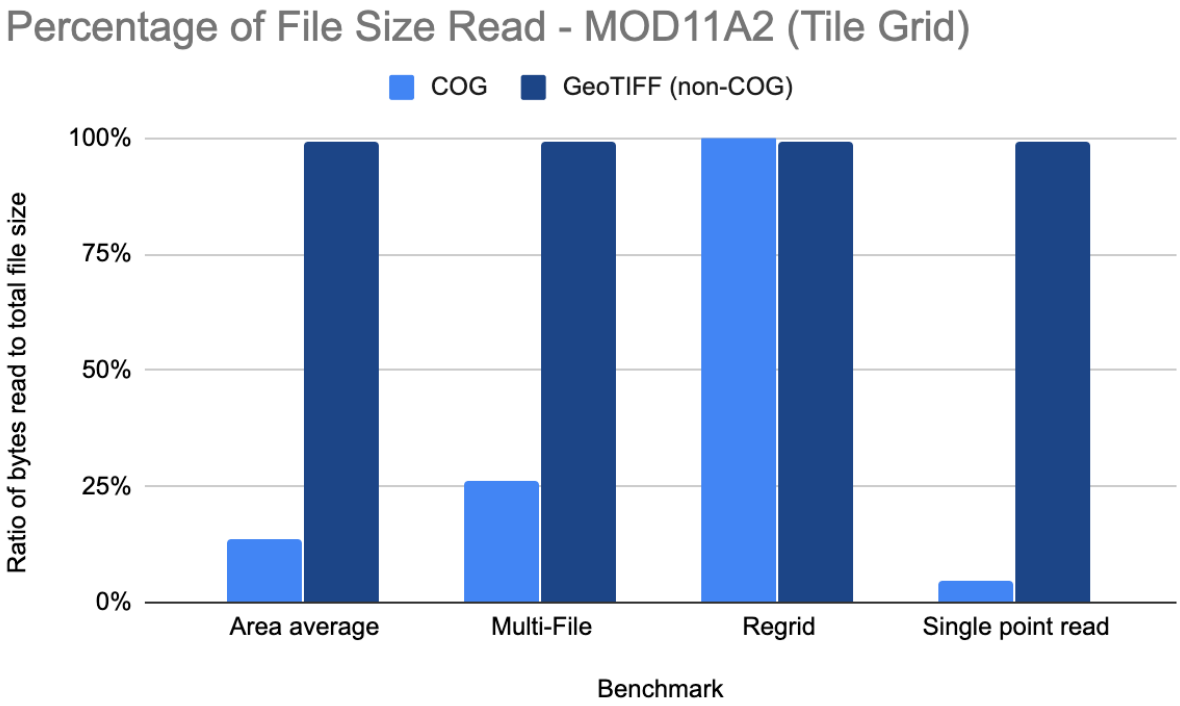




Note that for ATL03, we encountered errors in attempting to process data in situ which did not occur when downloading the data. The above chart represents a full download of the data, as would be necessary for end users to overcome this error.

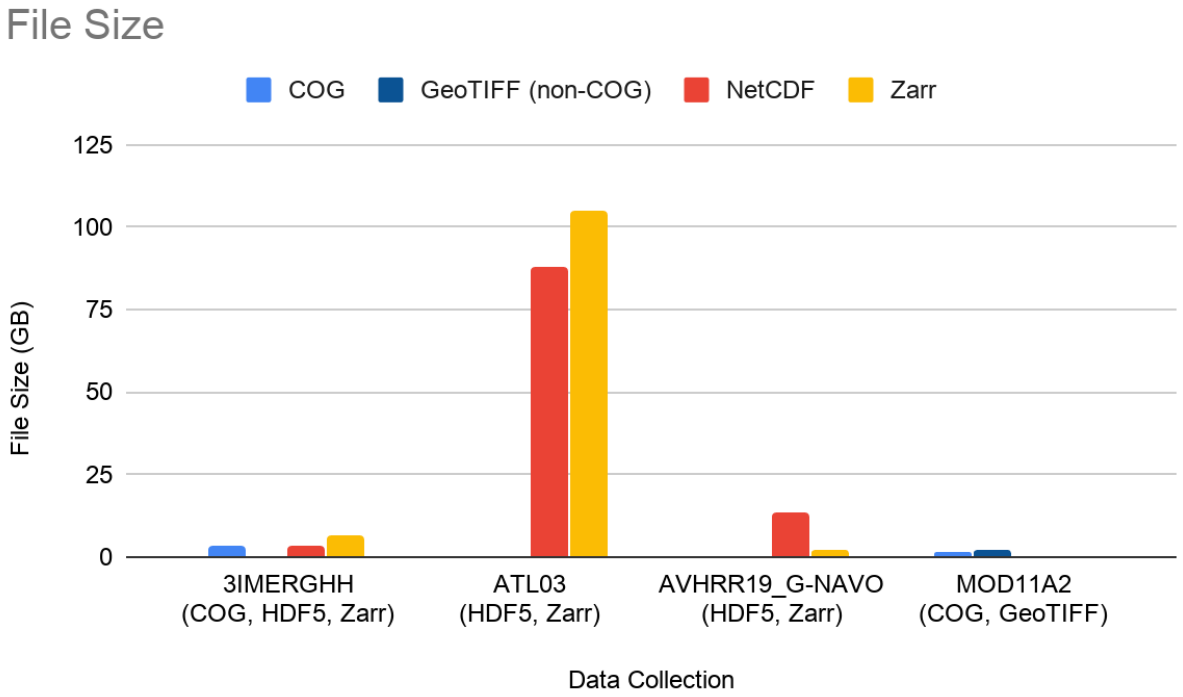


Note that for the single point read use case, Zarr is represented on the above chart but only accessed 0.13% of the total file size, causing it to not appear in that column.



File Size

The following chart represents the average file size using default transformation settings, with no attempt to alter or optimize compression.



When comparing file sizes, we again note the use of default compression settings. This resulted in a substantial increase in file size for Zarr for two collections and a dramatic decrease on a third collection. We recommend a more thorough analysis of the impact of compression on file size and performance, though this is generally an orthogonal concern to format, as most formats support multiple compression types.

Usability Findings During Benchmarking

We encountered the following challenges in making use of the data during tests:

Lack of GDAL Support

We would have liked to have tested operations run by GDAL. GDAL, however, does not yet have a Zarr driver. It further does not yet support reading NetCDF-4files from Amazon S3. Because of this, we chose not to make use of it for any format, despite it being a popular tool and providing much simpler and likely faster regridding operations than we ultimately used.

Errors in NetCDF-4/ HDF Libraries

We encountered the following errors while using NetCDF-4files:

- AVHRR19_G-NAVO: “RuntimeError: Unspecified error in H5DSiterate_scales (return value <0)” We encountered this cryptic error preventing us from opening any files when running on Amazon Linux (based on CentOS). We resolved it by downloading the HDF5 binaries and development headers and recompiling the h5py library from source.

- ATL03: Out of memory area during the shape based averaging benchmark. While we were able to successfully run the benchmark against some of the files, we were not able to run the complete benchmark against all of the ATL03 files. The EC2 instance we used had 7.5GB of memory, but for at least one of the test files we saw that all of the memory on the instance was used and the python benchmark script errored out.
- We encountered a memory leak when performing time series analysis on AVHRR19_G-NAVO, which caused our test process to abort as the system ran out of memory. Debugging this uncovered an efficiency gain we could apply to this test case across all formats, but the underlying leak still exists.

Necessary Performance Tuning for NetCDF-4/ HDF

NetCDF-4and HDF require tuning of xarray's block size parameter, which describes the minimum number of bytes to be read from S3 per call. The parameter is intended to allow sequences of reads of nearby data to avoid repeated network requests. The default value of this parameter, 5MB, caused pathologically bad behavior for NetCDF; doing an area average of a single parameter in a single variable in a single file egressed approximately ten times the number of bytes contained in the file and took two and a half minutes to perform. Users simply opening a remote file would encounter this behavior. We tuned the block size parameter, though optimal performance would require re-tuning it for each use. We found 5KB provided nearly optimal speed while keeping network traffic relatively low and used that for our tests. It is worth noting that the value for optimal performance is different from the value for optimal network usage (cost) and users can tune the parameter or fail to tune the parameter to the detriment of the archive.

Performance Influencing Usage Choices

While producing and debugging code for non-optimized formats, we found ourselves iterating on the code while using an optimized format and only using the non-optimized format when changes worked on the optimized format. The development experience changed dramatically when code could be run in a couple of seconds rather than tens of seconds. This would translate to easier algorithm iteration by end users.

Additional Considerations

While researching this study, we encountered additional factors to consider when choosing a format which do not lend themselves to a straightforward scoring approach. We capture them in this section.

File Packaging and Benchmarking

While choice of format has inherent performance implications, decisions made when packaging files can be highly impactful. Further, there is no one-size-fits-all packaging for data, as the optimal packaging is highly use-case dependent. In general, files should be arranged such that an operation can access the data of interest and only those data in either as few requests as possible (for typical bandwidth-constrained end-user machines) or several highly-parallel requests (for HPC and highly scalable systems). To do this, those packaging files should

maximize spatial locality of reference, i.e. arrange files such that bytes typically read together are in contiguous blocks in the file, and that any block or chunk sizing allows for reading those bytes in a small number of requests without extraneous data being read.

These packaging considerations constitute trade-offs that may favor some use cases over others. Optimizing for time-series analysis at a particular location may favor placing temporally proximate data close to one another on disk for each latitude and longitude with less concern for keeping nearby latitudes and longitudes close to one another on disk. Optimizing for spatial analysis at a particular time would do the opposite. Further, even optimizing for spatial access may produce greatly differing file structures depending on whether users typically have a small, large, or global area of interest; file structures allowing a large area to be loaded in a single read tend to require loading small areas to read significant extraneous data, while file structures minimizing extraneous reads for small areas tend to produce a high number of requests and subsequent manipulation when reading large areas.

Because of this variance in file structure, we recommend that, when performance and network access patterns are a concern, tests capturing typical use cases be run on a per-collection level to inform packaging decisions, if not format decisions.

Server Software

Server software, in particular OPeNDAP and HSDS / Kita, may alter the user-facing performance characteristics and tool support for some formats, particularly NetCDF-4, HDF5, and HDF in the Cloud. Both OPeNDAP and HSDS optimize for network-based partial reads and take advantage of parallelism in cloud environments. This effectively allows tools which often otherwise require whole-file downloads for NetCDF-4 and HDF5 to instead take advantage of partial file access done efficiently near the data.

While coupling the data to server software can make the underlying formats faster and easier for end-users to work with, it comes with increased operational complexity and potentially cost. In order for a user to gain the benefits of these formats, users must rely on an organization to operate instances of the server software, with appropriate scaling, monitoring, and maintenance for both the underlying software as well as (in many cases) the operating system running it. We do not attempt to characterize operational staffing costs or the cost of additional computing power other than noting that it is certainly present when relying on servers to mediate data access.

Alternately, users can reap the benefits of natively cloud-optimized formats by pointing libraries directly at the object stores which are already storing and serving cloud-based archives with no additional server software necessary.

For the purposes of this study, therefore, we separated the presence of such server software from the characteristics of the underlying format.

Other Formats

We note that, while the studied formats are among the most popular for storing and manipulating Earth observation data, other potentially beneficial formats exist. In this section, we note a few that may be situationally useful or otherwise worth watching.

JPEG 2000 with JP2 Georeferencing

JPEG 2000 is an ISO standard with an OGC standard for metadata georeferencing. It has a lossless compression mode using a wavelet-based method that can have different, often compelling compression ratio characteristics compared to the formats studied. The wavelet-based storage also inherently supports progressive rendering, allowing previews of data during access, efficient overview creation, and potentially lower bandwidth usage.

On the other hand, JPEG 2000's open source drivers have some defects, are approximately 10 times slower than proprietary alternatives, with read characteristics that are potentially problematic for cost in cloud environments.[30] The fast proprietary drivers, though, demonstrate that better performance is possible. It may be worthwhile to investigate the benefits of the format and whether the cost savings from compression gains would make an investment in improving the open source driver worthwhile.

Point Cloud Formats (Entwine Point Tile and others)

While most of the formats investigated are well-suited to array-based data, we note the existence of formats that are better suited to fast analysis of point cloud data. Because they are specialized to this particular use and tend to be somewhat niche, we did not include them in this study. An example here is Entwine Point Tile[31] which is a format that stores point clouds in an octree. This allows high spatial locality of reference while also enabling tools to visualize the point cloud at different levels of detail, two features which all formats we investigated would struggle to duplicate.

TileDB, Meta Raster Format (MRF), Cloud Raster Format, and Similar

We note the existence of several formats which are functionally very similar to Cloud-Optimized GeoTIFF, in that they are being optimized for access of multispectral data and are typically organized into tiles. TileDB[32], Meta Raster Format (MRF)[33], and ESRI Cloud Raster Format[34] are examples. While none of these formats have a level of adoption comparable to Cloud-Optimized GeoTIFF, the existence of several implementations optimizing access to multispectral data suggests that we should consider monitoring the format landscape in the future for new developments. It should also be noted that ESDIS has experience with MRF as part of the Global Image Browse Services (GIBS).

[30] Chris Tweedie, "JPEG 2000 Is Slow … Or Is It? ...," *Sensing Change Blog*, last modified December 28, 2015, accessed November 19, 2019, https://blog.hexagongeospatial.com/jpeg2000-quirks/.
[31] See https://entwine.io/entwine-point-tile.html
[32] https://tiledb.com/
[33] https://github.com/nasa-gibs/mrf
[34] https://pro.arcgis.com/en/pro-app/help/data/imagery/supported-raster-dataset-file-formats.htm#GUID-73EFF808-12B0-4434-B3C3-4BA8423B64FE

Conclusions

Overarching Recommendations

Regardless of selected format and compression algorithm, data providers should optimize files for partial access over HTTP using the Range header, which is the underlying mechanism used to retrieve data from cloud-based object stores but is also possible to use from a variety of systems, including on premises servers. Doing so can improve performance for services and end users, decrease costs and system demands, and simultaneously improve user experience and enable new access patterns for a given format and compression choice. Our quantitative benchmarks on GeoTIFF vs Cloud-Optimized GeoTIFF showed this to be a win-win situation having at worst comparable performance and cost profiles and in most cases dramatically improving both.

That all said, it is clear from this study that the choice of format and compression itself is critical for enabling archival integrity, broad tool support, and fast, cost-effective use. We find no one-size-fits-all solution but offer some broad guidelines to inform choices for specific needs.

Format Comparisons

We have provided a weighted matrix informing decision making about file formats for general use, with configurable weights based on desirable attributes. Because we do not supply these weights, we do not attempt to form a conclusion about a single best format choice. While some decisions seem relatively straightforward, like the choice of Cloud Optimized GeoTIFF over standard GeoTIFF when a TIFF is desired, other format decisions require more nuanced trade-offs. To help understand those trade-offs, we recognize a few themes that would make formats more or less appropriate for a particular use, summarized here.

Long-Term Archival Formats

Long-term archival use emphasizes the need for a stable, self-describing, standard format, preferably with multiple reader implementations, which can be checked for integrity. It also emphasizes optimal disk usage over performance. Finally, it relies heavily on portability, i.e. the ability to move files between different storage options and access them in places other than their original storage location. The combined needs for portability and integrity checking tend to favor formats in which files are self-contained rather than spread between objects in an object store, as unitary files can be readily and atomically moved, checksummed as a whole, and are typically easier to conceptualize. COG, GeoTIFF, and NetCDF-4are forerunners here, with the latter having an edge in being able to accommodate more metadata and more different types of data. Zarr is particularly weak in this category, as its format is not recognized by any standards body and may evolve over time, and a single Zarr file tends to be spread across objects in an object store, weakening direct portability via copying objects (though notably the Zarr library supports storage backends such as zip that are individual files).

Multispectral Data Formats

Cloud Optimized GeoTIFF specializes in structuring and displaying imagery (or, layers of two dimensional arrays) in a single file. It has a wide range of tooling, both for its Cloud Optimized variant as well as the GeoTIFF and TIFF standards it is based on. It has very few down-sides

as both an archival and analysis-ready format for this type of data, the most noteworthy of which is its reliance on sidecar files for capturing large amounts of metadata, which slightly compromises its self-describability and ease of archival.

Multidimensional Array Data Formats

NetCDF, Zarr, and HDF in the Cloud all natively store hierarchical multidimensional array data, making them appropriate for data with more than two dimensions. Of these, Zarr stands out for an analysis-ready format for the Cloud, as it provides performance optimized for a network environment in a way that requires no specialized server software. NetCDF-4lacks some of the network optimization, while HDF in the Cloud requires specialized server software. See the "Long-Term Archival Formats" section above for reservations about Zarr as an archival format.

Specialized Data Formats

Parquet has potentially strong uses for particular data types and particular types of analysis on common data types. Among the formats studied, it is uniquely suited to database- or spreadsheet-style data. It also has fast yet powerful query capabilities with SQL-like interfaces, some of which, like Amazon Athena, are serverless and operate at scale. These interfaces make it very useful for analysis requiring simple mathematical operations such as arithmetic, minimums, maximums, and averages. The fact that it is not structured to support array-based data, while a weakness in dealing with spatial operations, can be a strength in providing even performance across both spatial and time series analysis for the same dataset.

The strength of Parquet in these circumstances underscores a salient conclusion: it is important to match the file format and internal data structure to the most important usage patterns. These usage patterns may differ between products and there may be multiple important usage patterns for a single product. As such, the ability of an organization to meet their user's needs may be less about finding the singular optimal data format and more about the organization's ability to stage select, highly used data in multiple different formats simultaneously.

Recommended Follow-On Work

This study constitutes an overview of some available network-optimized file formats with benchmarks against a handful of disparate datasets using consistent conversion, compression, and representation settings. We note several potential lines of study which are not in scope for this effort but nonetheless may aid in decision making about file formats:

1. Operational trials against real data for select users. Provide and publicize an alternative access format to users of an existing dataset. Use metrics to determine the actual cost of storage, compute, and egress against their real use cases and quantify usability by allowing users to vote with their feet. Reach out to other organizations and entities that may have such metrics.
2. Total cost analysis of formats and of in-cloud services vs non-cloud partial file access. The study data contains significant useful information in looking at the total cost to provide data for given use. To provide a better cost analysis, we would need to look at operational costs of maintaining server software and egress costs in delivering service responses to users.
3. Store collections as a single logical file, in Zarr and NetCDF-4in particular, and investigate the impact on ingest time and benchmarks. Zarr allows appending time data or ranges of time data without overwriting the dataset. NetCDF-4allows this as well but the underlying object store would require overwriting the entire dataset. This change in representation would likely improve time series performance while potentially negatively impacting fitness for archival use, spatial performance, and ability to search and select temporal data from non-temporal metadata.
4. Repeat the study, focusing on server software that enables faster access across a wide range of tools instead of focusing on formats. OPeNDAP and The HDF Group's Kita software both provide compelling cases for their use but ought to be examined for operational cost, scalability, and performance compared to simpler access methods.

Additionally, the analysis may be extended with more use cases, more data formats, and more file structure optimizations of original and network-optimized formats.

Appendix A: Acronym List

API - Application Programming Interface
ArcGIS - Aeronautical Reconnaissance Coverage Geographic Information System
CF - Climate and Forecast Metadata Conventions
COG - Cloud Optimized GeoTIFF
EOSDIS - Earth Observing System Data and Information System
GDAL - Geospatial Data Abstraction Library
GeoTIFF - Georeferenced Tagged Image File Format
GIBS - Global Image Browse Services
HDF - Hierarchical Data Format
HDFS - Hadoop Distributed File System
HPC - High Performance Computing
HSDS - Highly Scalable Data Service
HTTP - Hypertext Transfer Protocol
JP2 - JPEG 2000
JPEG - Joint Photographic Experts Group
JSON - JavaScript Object Notation
ISO - International Standards Organization
MRF - Meta Raster Format
NCO - NetCDF-4Operators
NetCDF-4- Network Common Data Form
OGC - Open Geospatial Consortium
OPeNDAP - Open-source Project for a Network Data Access Protocol
PAM - Persistent Auxiliary Metadata
QGIS - Quantum Geographic Information System
TIFF - Tagged Image File Format
W3C - World Wide Web Consortium
WMS - Web Mapping Services

Appendix B - Scoring Justification and Notes

Additional details regarding how scores were determined can be found at the spreadsheet below. It also contains raw data from the quantitative analysis results:

https://docs.google.com/spreadsheets/d/11mCu6-Ql7LBnoIPk29yJcNoFKi-CdlOrUTLhyU5e5dc/edit?usp=sharing